

Лабораторна робота №5

ІПЗ-21-5 Богайчук Денис

Мета: використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити методи ансамблів у машинному навчанні.

Завдання 1. Створення класифікаторів на основі випадкових та гранично випадкових лісів

Лістинг коду програми

```
import argparse

import matplotlib.pyplot as plt
import numpy as np
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split

from utilities import visualize_classifier

# Argument parser
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Classify data using Ensemble Learning techniques')
    parser.add_argument('--classifier-type', dest='classifier_type',
                        required=True, choices=['rf', 'erf'],
                        help="Type of classifier to use; can be either 'rf' or 'erf'")
    return parser

if __name__ == '__main__':
    # Parse the input arguments
    args = build_arg_parser().parse_args()
    classifier_type = args.classifier_type

    # Load input data
    input_file = 'Lab5/data_random_forests.txt'
    data = np.loadtxt(input_file, delimiter=',')
    X, y = data[:, :-1], data[:, -1]

    # Separate input data into three classes based on labels
    class_0 = np.array(X[y == 0])
    class_1 = np.array(X[y == 1])
    class_2 = np.array(X[y == 2])

    # Visualize input data
    plt.figure()
    plt.scatter(class_0[:, 0], class_0[:, 1], s=75, facecolors='white',
                edgecolors='black', linewidth=1, marker='s')
```

```

plt.scatter(class_1[:, 0], class_1[:, 1], s=75, facecolors='white',
            edgecolors='black', linewidth=1, marker='o')
plt.scatter(class_2[:, 0], class_2[:, 1], s=75, facecolors='white',
            edgecolors='black', linewidth=1, marker='^')
plt.title('Input data')

# Split data into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=5)

# Ensemble Learning classifier
params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}
if classifier_type == 'rf':
    classifier = RandomForestClassifier(**params)
else:
    classifier = ExtraTreesClassifier(**params)

classifier.fit(X_train, y_train)
visualize_classifier(classifier, X_train, y_train, 'Training dataset')

y_test_pred = classifier.predict(X_test)
visualize_classifier(classifier, X_test, y_test, 'Test dataset')

# Evaluate classifier performance
class_names = ['Class-0', 'Class-1', 'Class-2']
print("\n" + "#" * 40)
print("\nClassifier performance on training dataset\n")
print(classification_report(y_train, classifier.predict(X_train),
target_names=class_names))
print("#" * 40 + "\n")

print("#" * 40)
print("\nClassifier performance on test dataset\n")
print(classification_report(y_test, y_test_pred, target_names=class_names))
print("#" * 40 + "\n")

# Compute confidence
test_datapoints = np.array([[5, 5], [3, 6], [6, 4], [7, 2], [4, 4], [5, 2]])

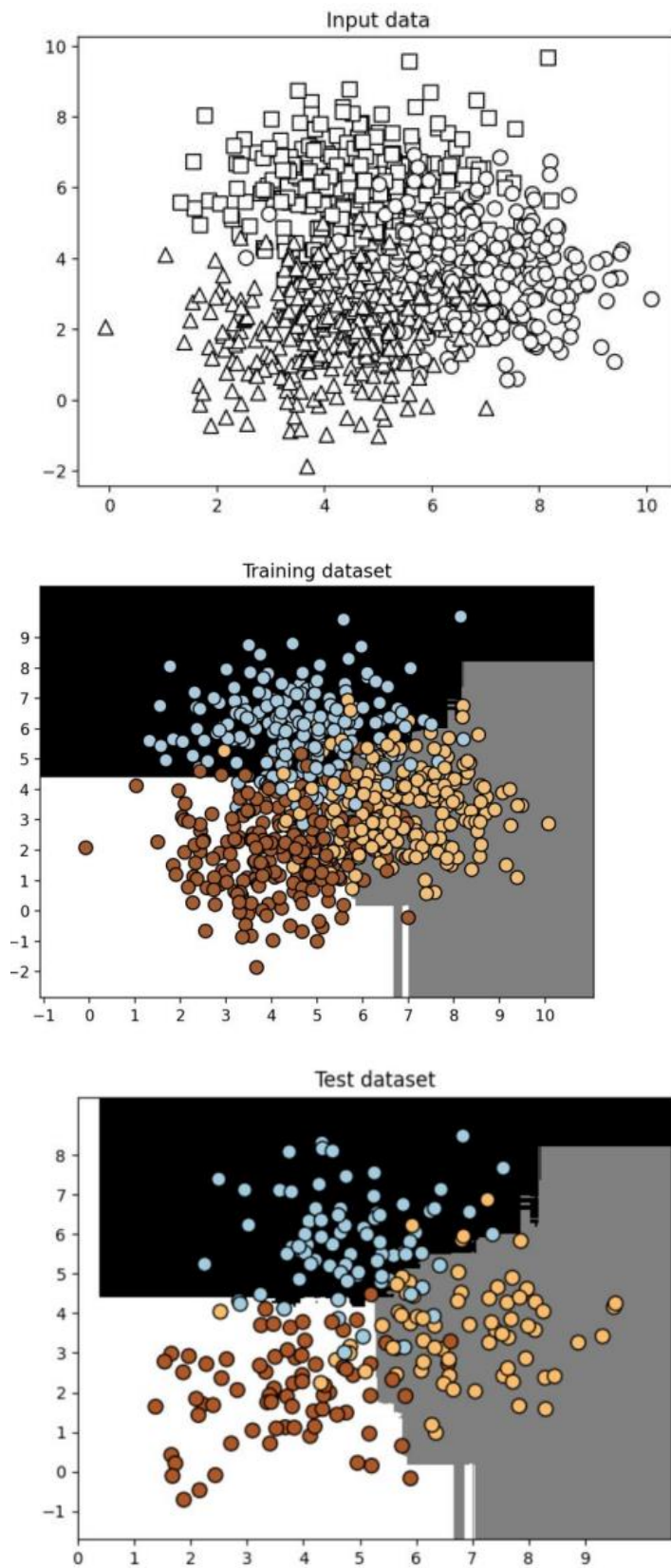
print("\nConfidence measure:")
for datapoint in test_datapoints:
    probabilities = classifier.predict_proba([datapoint])[0]
    predicted_class = 'Class-' + str(np.argmax(probabilities))
    print('\nDatapoint:', datapoint)
    print('Predicted class:', predicted_class)

# Visualize the datapoints
visualize_classifier(classifier, test_datapoints, [0] *
len(test_datapoints), 'Test datapoints')

plt.show()

```

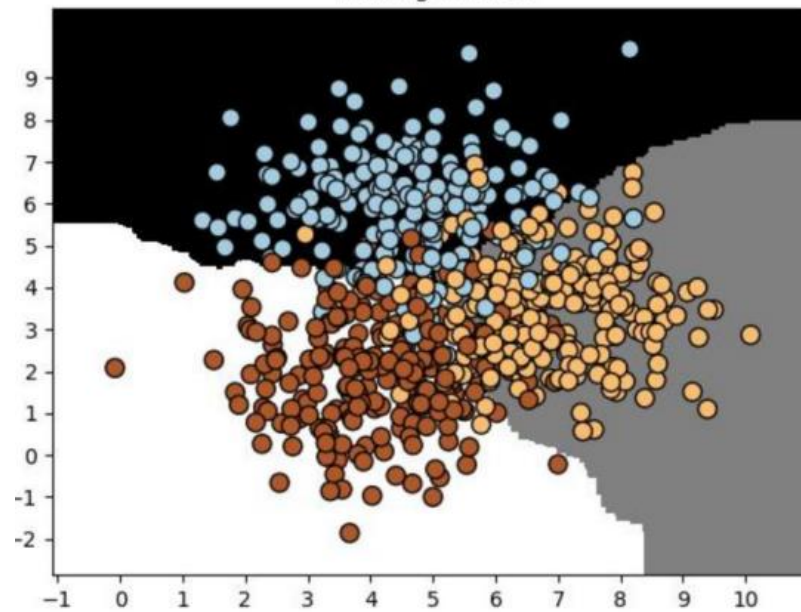
Результат виконання коду програми



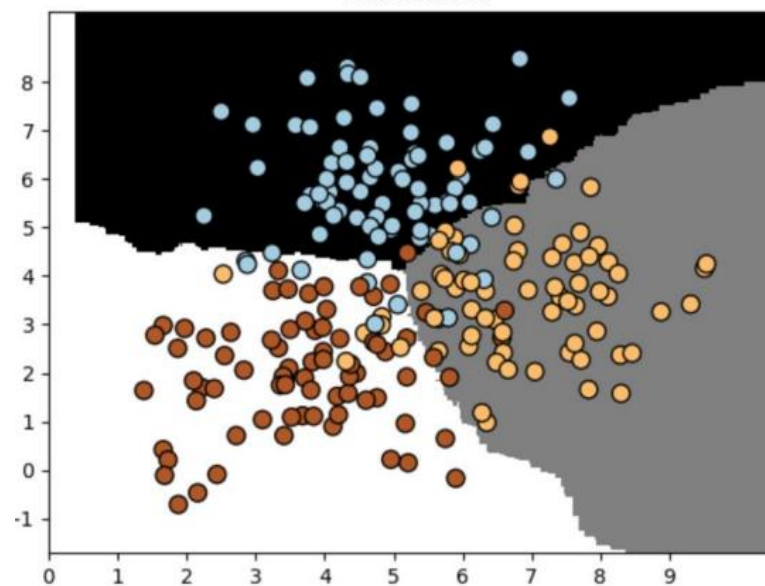
Class-0	0.92	0.85	0.88	79
Class-1	0.86	0.84	0.85	70
Class-2	0.84	0.92	0.88	76
accuracy			0.87	225
macro avg	0.87	0.87	0.87	225
weighted avg	0.87	0.87	0.87	225

#####

Training dataset

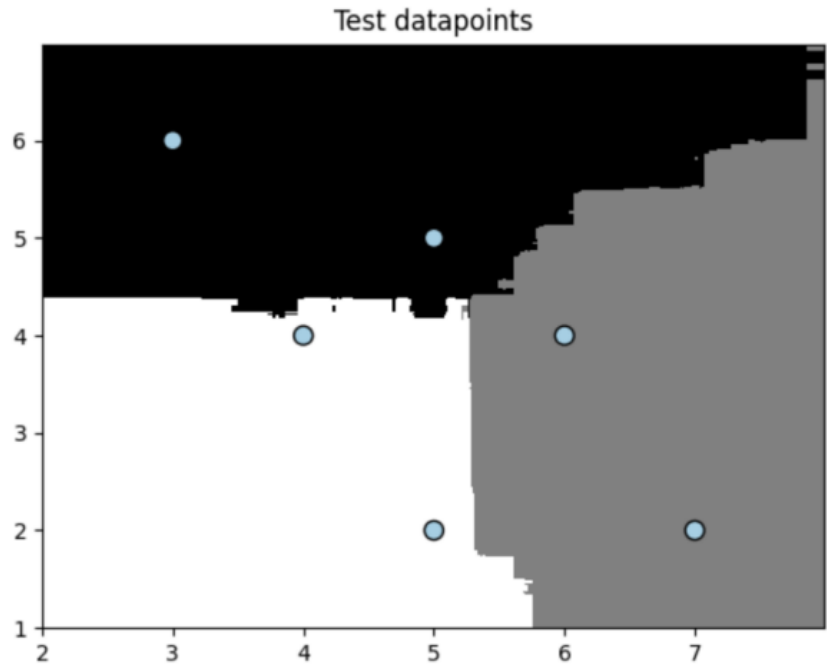


Test dataset



Classifier performance on test dataset

	precision	recall	f1-score	support
Class-0	0.92	0.85	0.88	79
Class-1	0.84	0.84	0.84	70
Class-2	0.85	0.92	0.89	76
accuracy			0.87	225
macro avg	0.87	0.87	0.87	225
weighted avg	0.87	0.87	0.87	225



Confidence measure:

Datapoint: [5 5]

Predicted class: Class-0

Datapoint: [3 6]

Predicted class: Class-0

Datapoint: [6 4]

Predicted class: Class-1

Datapoint: [7 2]

Predicted class: Class-1

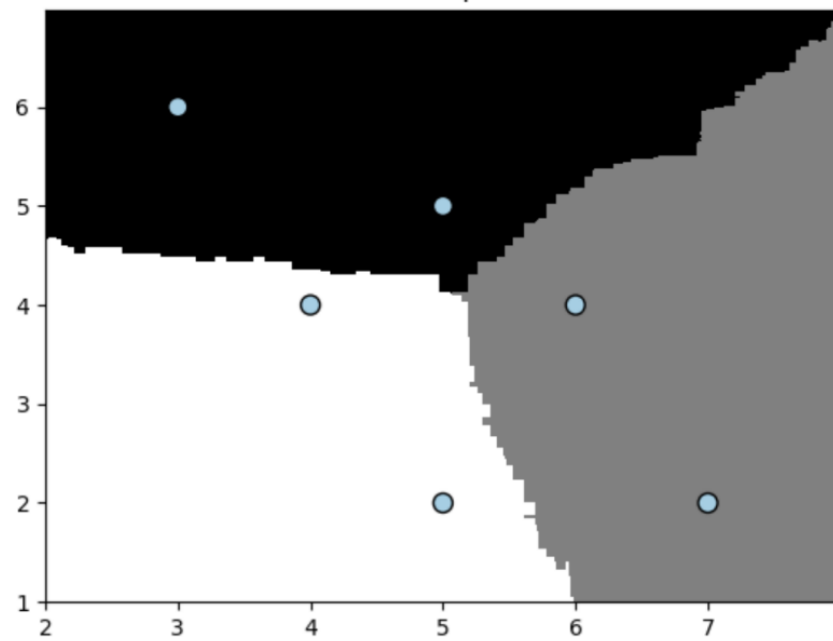
Datapoint: [4 4]

Predicted class: Class-2

Datapoint: [5 2]

Predicted class: Class-2

Test datapoints



```
Confidence measure:

Datapoint: [5 5]
Predicted class: Class-0

Datapoint: [3 6]
Predicted class: Class-0

Datapoint: [6 4]
Predicted class: Class-1

Datapoint: [7 2]
Predicted class: Class-1

Datapoint: [4 4]
Predicted class: Class-2

Datapoint: [5 2]
Predicted class: Class-2
```

Завдання 2. Обробка дисбалансу класів.

Лістинг виконання коду програми

```
import sys

import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

from utilities import visualize_classifier

# Завантаження вхідних даних
input_file = 'data_imbalance.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]

# Поділ вхідних даних на два класи на підставі міток
```

```

class_0 = np.array(X[y == 0])
class_1 = np.array(X[y == 1])

# Візуалізація вхідних даних
plt.figure()
plt.scatter(class_0[:, 0], class_0[:, 1], s=75, facecolors='black',
            edgecolors='black', linewidth=1, marker='x')
plt.scatter(class_1[:, 0], class_1[:, 1], s=75, facecolors='white',
            edgecolors='black', linewidth=1, marker='o')
plt.title('Input data')

# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=5)

# Класифікатор на основі гранично випадкових лісів
params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}
if len(sys.argv) > 1:
    if sys.argv[1] == 'balance':
        params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0,
            'class_weight': 'balanced'}
    else:
        raise TypeError("Invalid input argument; should be 'balance'")

classifier = ExtraTreesClassifier(**params)
classifier.fit(X_train, y_train)
visualize_classifier(classifier, X_train, y_train, 'Training dataset')

y_test_pred = classifier.predict(X_test)
visualize_classifier(classifier, X_test, y_test, 'Test dataset')

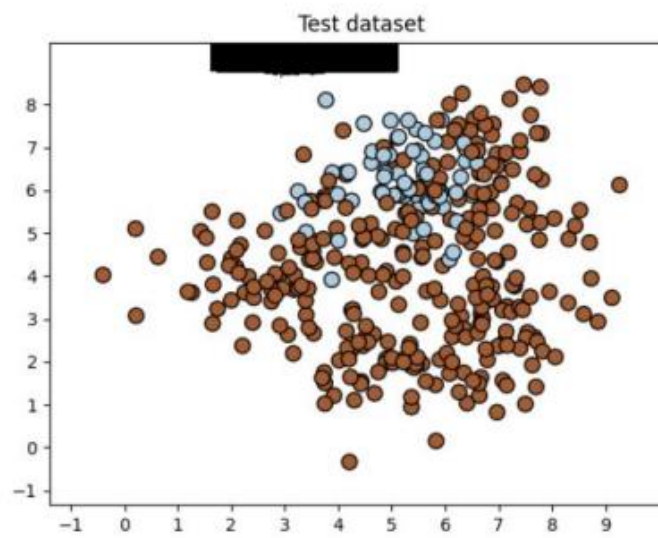
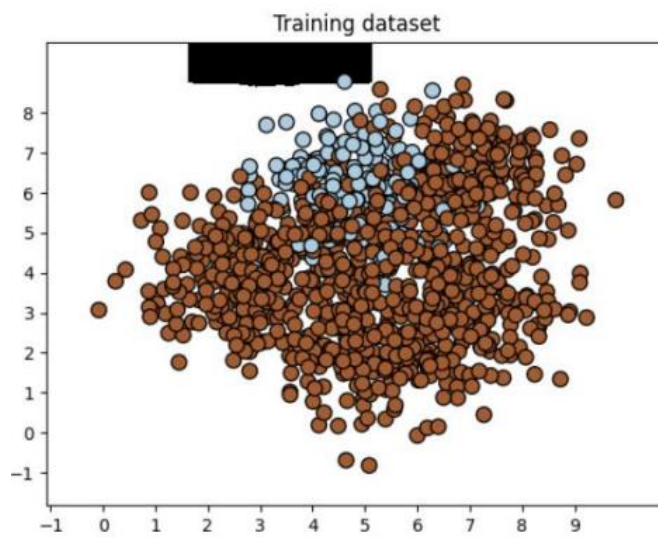
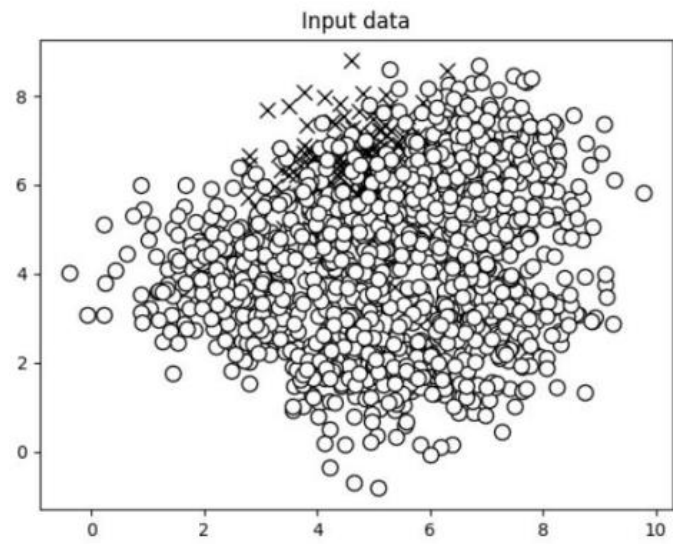
# Обчислення показників ефективності класифікатора
class_names = ['Class-0', 'Class-1']
print("\n" + "#" * 40)
print("\nClassifier performance on training dataset\n")
print(classification_report(y_train, classifier.predict(X_train),
    target_names=class_names))
print("#" * 40 + "\n")

print("#" * 40)
print("\nClassifier performance on test dataset\n")
print(classification_report(y_test, y_test_pred, target_names=class_names))
print("#" * 40 + "\n")

plt.show()

```


Результат виконання коду програми



```

Classifier performance on training dataset

              precision    recall  f1-score   support

   Class-0       1.00        0.01    0.01        181
   Class-1       0.84        1.00    0.91        944

 accuracy              0.84        1125
 macro avg           0.92        0.50    0.46        1125
 weighted avg        0.87        0.84    0.77        1125

#####

#####

Classifier performance on test dataset

              precision    recall  f1-score   support

   Class-0       0.00        0.00    0.00         69
   Class-1       0.82        1.00    0.90        306

 accuracy              0.82        375
 macro avg           0.41        0.50    0.45        375
 weighted avg        0.67        0.82    0.73        375

#####

```

Завдання 3. Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку.

Лістинг коду програми

```

import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import pandas as pd

from utilities import visualize_classifier

input_file = 'data_random_forests.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]

# Розбиття даних на три класи на підставі міток
class_0 = np.array(X[y == 0])
class_1 = np.array(X[y == 1])
class_2 = np.array(X[y == 2])

```

```

# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=5)

# Визначення сітки значень параметрів
parameter_grid = [{'n_estimators': [100], 'max_depth': [2, 4, 7, 12, 16]},
                  {'max_depth': [4], 'n_estimators': [25, 50, 100, 250]}
                  ]

metrics = ['precision_weighted', 'recall_weighted']

for metric in metrics:
    print("\n#### Searching optimal parameters for", metric)

    classifier = GridSearchCV(
        ExtraTreesClassifier(random_state=0),
        parameter_grid, cv=5, scoring=metric)
    classifier.fit(X_train, y_train)

    df = pd.DataFrame(classifier.cv_results_)
    df_columns_to_print = [column for column in df.columns if 'param' in column
    or 'recall' in column]
    print(df[df_columns_to_print])

    print("\nBest parameters:", classifier.best_params_)

    y_pred = classifier.predict(X_test)
    print("\nPerformance report:\n")
    print(classification_report(y_test, y_pred))

```

Результат виконання коду програми

```

#### Searching optimal parameters for precision_weighted

```

	param_max_depth	param_n_estimators	params
0	2	100	{'max_depth': 2, 'n_estimators': 100}
1	4	100	{'max_depth': 4, 'n_estimators': 100}
2	7	100	{'max_depth': 7, 'n_estimators': 100}
3	12	100	{'max_depth': 12, 'n_estimators': 100}
4	16	100	{'max_depth': 16, 'n_estimators': 100}
5	4	25	{'max_depth': 4, 'n_estimators': 25}
6	4	50	{'max_depth': 4, 'n_estimators': 50}
7	4	100	{'max_depth': 4, 'n_estimators': 100}
8	4	250	{'max_depth': 4, 'n_estimators': 250}

```

Best parameters: {'max_depth': 2, 'n_estimators': 100}

Performance report:

```

	precision	recall	f1-score	support
0.0	0.94	0.81	0.87	79
1.0	0.81	0.86	0.83	70
2.0	0.83	0.91	0.87	76
accuracy			0.86	225
macro avg	0.86	0.86	0.86	225
weighted avg	0.86	0.86	0.86	225

```
##### Searching optimal parameters for recall_weighted
```

	param_max_depth	param_n_estimators	params
0	2	100	{'max_depth': 2, 'n_estimators': 100}
1	4	100	{'max_depth': 4, 'n_estimators': 100}
2	7	100	{'max_depth': 7, 'n_estimators': 100}
3	12	100	{'max_depth': 12, 'n_estimators': 100}
4	16	100	{'max_depth': 16, 'n_estimators': 100}
5	4	25	{'max_depth': 4, 'n_estimators': 25}
6	4	50	{'max_depth': 4, 'n_estimators': 50}
7	4	100	{'max_depth': 4, 'n_estimators': 100}
8	4	250	{'max_depth': 4, 'n_estimators': 250}

Best parameters: {'max_depth': 2, 'n_estimators': 100}

Performance report:

	precision	recall	f1-score	support
0.0	0.94	0.81	0.87	79
1.0	0.81	0.86	0.83	70
2.0	0.83	0.91	0.87	76
accuracy			0.86	225
macro avg	0.86	0.86	0.86	225
weighted avg	0.86	0.86	0.86	225

```
##### Searching optimal parameters for precision_weighted
```

	param_max_depth	param_n_estimators	params
0	2	100	{'max_depth': 2, 'n_estimators': 100}
1	4	100	{'max_depth': 4, 'n_estimators': 100}
2	7	100	{'max_depth': 7, 'n_estimators': 100}
3	12	100	{'max_depth': 12, 'n_estimators': 100}
4	16	100	{'max_depth': 16, 'n_estimators': 100}
5	4	25	{'max_depth': 4, 'n_estimators': 25}
6	4	50	{'max_depth': 4, 'n_estimators': 50}
7	4	100	{'max_depth': 4, 'n_estimators': 100}
8	4	250	{'max_depth': 4, 'n_estimators': 250}

Best parameters: {'max_depth': 2, 'n_estimators': 100}

Performance report:

	precision	recall	f1-score	support
0.0	0.94	0.81	0.87	79
1.0	0.81	0.86	0.83	70
2.0	0.83	0.91	0.87	76
accuracy			0.86	225
macro avg	0.86	0.86	0.86	225
weighted avg	0.86	0.86	0.86	225

```
##### Searching optimal parameters for recall_weighted
```

	param_max_depth	param_n_estimators	params
0	2	100	{'max_depth': 2, 'n_estimators': 100}
1	4	100	{'max_depth': 4, 'n_estimators': 100}
2	7	100	{'max_depth': 7, 'n_estimators': 100}
3	12	100	{'max_depth': 12, 'n_estimators': 100}
4	16	100	{'max_depth': 16, 'n_estimators': 100}
5	4	25	{'max_depth': 4, 'n_estimators': 25}
6	4	50	{'max_depth': 4, 'n_estimators': 50}
7	4	100	{'max_depth': 4, 'n_estimators': 100}
8	4	250	{'max_depth': 4, 'n_estimators': 250}

Best parameters: {'max_depth': 2, 'n_estimators': 100}

Performance report:

	precision	recall	f1-score	support
0.0	0.94	0.81	0.87	79
1.0	0.81	0.86	0.83	70
2.0	0.83	0.91	0.87	76
accuracy			0.86	225
macro avg	0.86	0.86	0.86	225
weighted avg	0.86	0.86	0.86	225

Для метрики recall було отримано іншу комбінацію параметрів, що цілком логічно, оскільки точність і recall відображають різні аспекти оцінки моделі й потребують різних налаштувань.

Завдання 4. Обчислення відносної важливості ознак.

Лістинг коду програми

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.datasets import fetch_california_housing
from sklearn.metrics import mean_squared_error, explained_variance_score
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle

# Завантаження даних із цінами на нерухомість
housing_data = fetch_california_housing()

# Перемішування даних
X, y = shuffle(housing_data.data, housing_data.target, random_state=7)

# Розбиття даних на навчальний та тестовий набори
```

```

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=7)

# Модель на основі регресора AdaBoost
regressor = AdaBoostRegressor(DecisionTreeRegressor(max_depth=4),
                               n_estimators=400, random_state=7)
regressor.fit(X_train, y_train)

# Обчислення показників ефективності регресора AdaBoost
y_pred = regressor.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
evs = explained_variance_score(y_test, y_pred)
print("\nADABOOST REGRESSOR")
print("Mean squared error =", round(mse, 2))
print("Explained variance score =", round(evs, 2))

# Вилучення важливості ознак
feature_importances = regressor.feature_importances_
feature_names = housing_data.feature_names

# Нормалізація значень важливості ознак
feature_importances = 100.0 * (feature_importances / max(feature_importances))

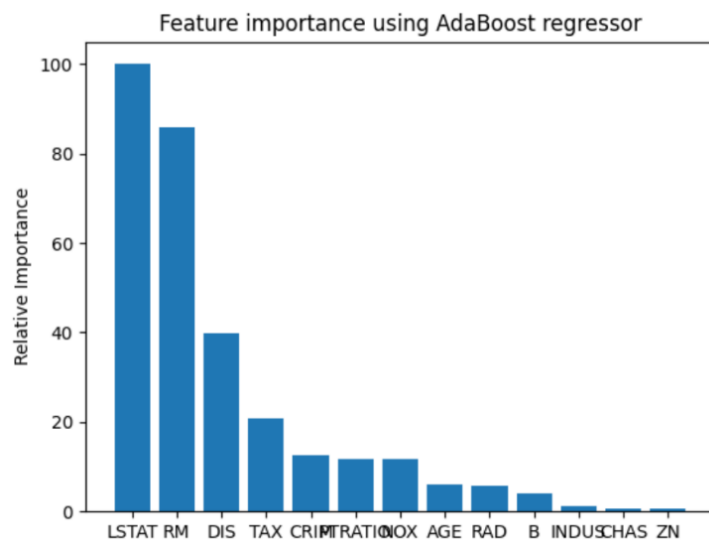
# Сортуння та перестановка значень
index_sorted = np.flipud(np.argsort(feature_importances))

# Розміщення міток уздовж осі X
pos = np.arange(index_sorted.shape[0]) + 0.5

# Побудова стовпчастої діаграми
plt.figure()
plt.bar(pos, feature_importances[index_sorted], align='center')
plt.xticks(pos, feature_names[index_sorted])
plt.ylabel('Relative Importance')
plt.title('Feature importance using AdaBoost regressor')
plt.show()

```

Результат виконання коду програми



ADABOOST REGRESSOR

Mean squared error = 1.18

Explained variance score = 0.47

Аналіз показує, що функція LSTAT має найбільший вплив у цьому наборі даних.

Завдання 5. Прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів.

Лістинг коду програми

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, mean_absolute_error
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.metrics import classification_report

# Завантаження вхідних даних
input_file = 'traffic_data.txt'
data = []
with open(input_file, 'r') as f:
    for line in f.readlines():
        items = line[:-1].split(',')
        data.append(items)

data = np.array(data)

# Перетворення рядкових даних на числові
label_encoder = []
X_encoded = np.empty(data.shape)
for i, item in enumerate(data[0]):
    if item.isdigit():
        X_encoded[:, i] = data[:, i]
    else:
        label_encoder.append(preprocessing.LabelEncoder())
        X_encoded[:, i] = label_encoder[-1].fit_transform(data[:, i])

X = X_encoded[:, :-1].astype(int)
y = X_encoded[:, -1].astype(int)

# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=5)

# Регресор на основі гранично випадкових лісів
params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}
regressor = ExtraTreesRegressor(**params)
regressor.fit(X_train, y_train)
```

```
# Обчислення характеристик ефективності регресора на тестових даних
y_pred = regressor.predict(X_test)
print("Mean absolute error:", round(mean_absolute_error(y_test, y_pred), 2))

# Тестування кодування на одиночному прикладі
test_datapoint = ['Saturday', '10:20', 'Atlanta', 'no']
test_datapoint_encoded = [-1] * len(test_datapoint)
count = 0
for i, item in enumerate(test_datapoint):
    if item.isdigit():
        test_datapoint_encoded[i] = int(test_datapoint[i])
    else:
        test_datapoint_encoded[i] =
int(label_encoder[count].transform([test_datapoint[i]]))
        count = count + 1
test_datapoint_encoded = np.array(test_datapoint_encoded)

# Прогнозування результату для тестової точки даних
print("Predicted traffic:", int(regressor.predict([test_datapoint_encoded])[0]))
```

Результат виконання коду програми

```
Mean absolute error: 7.42
Predicted traffic: 26
```

Висновок: У ході дослідження методів ансамблів у машинному навчанні за допомогою спеціалізованих бібліотек на мові Python було успішно реалізовано та проаналізовано різні підходи. Це дозволило оцінити їхню ефективність і зрозуміти, як ансамблеві методи підвищують точність і стійкість моделей машинного навчання.